Dokumentacja - "Refleks"

Skład grupy (G02, czwartek 16:15):
Jakub Sikora 2137 (przywódca gildi *"Walecznych DevOps'ów"*)
Patryk Krawczyk 251563
Adrian Jagieła ???

Systemy Wbudowane 2025

1. Opis płytki rozwojowej z mikrokontrolerem LPC1343

Charakterystyka ogólna

LPC1343 to 32-bitowy mikrokontroler firmy NXP z rdzeniem ARM Cortex-M3, przeznaczony do zastosowań wbudowanych wymagających wysokiej integracji, niskiego poboru mocy i bogatego zestawu peryferiów. Płytki rozwojowe, takie jak LPC-P1343 czy LPC-H1343, umożliwiają łatwe prototypowanie i testowanie aplikacji dzięki wyprowadzeniom wszystkich linii I/O oraz obecności licznych interfejsów sprzętowych.

Główne cechy mikrokontrolera LPC1343

- **Rdzeń:** ARM Cortex-M3, 32-bitowy RISC, taktowanie do 72 MHz.
- **Pamięć:** 32 kB Flash (programowa), 8 kB SRAM.
- **Zasilanie:** 2,0–3,6 V (typowo 3,3 V).
- **Obudowa:** LQFP48.
- **Wejścia/wyjścia:** 42 linie GPIO z konfigurowalnymi rezystorami podciągającymi-/ściągającymi oraz trybem open-drain.
- **Interfejsy komunikacyjne:**
  - USB 2.0 full-speed device z wbudowanym PHY.
  - UART z obsługą RS-485, FIFO i generacją ułamkowych prędkości transmisji.
  - SSP (SPI) z FIFO i obsługą wielu protokołów.
  - I2C z obsługą Fast-mode Plus (do 1 Mbit/s), trybem monitorowania i rozpoznawaniem wielu adresów.
- **Timer i liczniki:** 4 uniwersalne timery/liczniki (łącznie 4 wejścia capture, 13 wyjść match).
- **ADC:** 10-bitowy przetwornik analogowo-cyfrowy (do 8 kanałów).
- **Inne peryferia:**
  - Watchdog Timer (WDT).
  - Wbudowany bootloader obsługujący ISP i IAP.
  - NVIC (Nested Vectored Interrupt Controller).

Wyposażenie płytki rozwojowej (np. LPC-P1343, LPC-H1343)

- **Zasilanie:** układ zasilania z diodą sygnalizującą zasilanie (Power-on LED); możliwość zasilania przez USB lub zewnętrzne źródło (5–9 V).
- **USB:** złącze USB typu B do komunikacji i zasilania.
- **Debugowanie:** interfejs SWD (Serial Wire Debug) do programowania i debugowania.

- **Przyciski:** dwa przyciski użytkownika oraz przycisk RESET.
- **LEDy:** osiem diod LED użytkownika do testów i sygnalizacji.
- **Obszar prototypowy:** pole lutownicze do własnych rozszerzeń.
- **UEXT:** uniwersalne złącze rozszerzeń do podłączania dodatkowych modułów (np. czujników, wyświetlaczy).
- **Wyprowadzenia:** wszystkie porty mikrokontrolera wyprowadzone na złącza gold-pin.
- **Wymiary płytki:** typowo 80 x 50 mm (LPC-P1343), 38 x 38 mm (LPC-H1343).
- **Płytka:** FR-4, grubość 1,5 mm, czerwony lub zielony soldermask, biały nadruk elementów.

Schemat blokowy i układ zasilania

- **Zegar:** kwarc 12 MHz podłączony do pinów XTALIN/XTALOUT.
- **Zworki:** umożliwiają wybór źródła zasilania (3,3 V dla rdzenia i I/O), podłączenie linii SWO do debuggera, połączenie masy z wybranymi pinami.
- **Reset:** układ resetu z rezystorem 10 kΩ i przyciskiem.
- **Pobór prądu:** typowo ok. 20 mA.

Bezpieczeństwo i użytkowanie

- Płytka dostarczana jest w opakowaniu antystatycznym – należy unikać ładunków elektrostatycznych podczas pracy.
- Do programowania wymagany jest odpowiedni programator/debugger (np. ARM-JTAG-EW, ARM-USB-OCD, SWD).
- Zalecane środowiska programistyczne to IAR for ARM, Rowley CrossWorks lub dedykowane IDE Eclipse-based (np. LPCXpresso).

2. Podział pracy w zespole wraz z listą wykorzystanych funkcjonalności

Tabela 1. Procentowy podział pracy członków zespołu.

Imię i nazwisko	Udział (%)
Jakub Sikora	2
Adrian Jagieła	49
Patryk Krawczyk	49

Tabela 2. Lista wykorzystanych funkcjonalności i odpowiedzialnych za nie autorów.

Funkcjonalność	Autor
Timer	Jakub Sikora
Dźwięk	Jakub Sikora
Joystick (GPIO)	Jakub Sikora
Wyświetlacz OLED	Jakub Sikora
Interfejs SPI	Jakub Sikora
Linijka diod	Jakub Sikora
Interfejs I2C	Jakub Sikora
Moduł EEPROM	Jakub Sikora
Akcelerometr	Jakub Sikora
Czujnik oświetlenia	Jakub Sikora

3. Opis działania programu - instrukcja użytkownika

Gra toczy się przez pięć rund oraz rozpoczyna się od wyświetlenia ekranu powitalnego wraz z bieżącym rekordowym wynikiem uzyskanym w grze w trakcie uprzednich rozgrywek. Aktualny postęp gry można obserwować na pasku diod LED. Pierwsza runda rozpoczyna się po naciśnięciu joysticka przez użytkownika. Jest wtedy inicjowane trzysekundowe odliczanie wizualizowane na wyświetlaczu. Następnie, po pseudolosowym odstępie czasu, między jedną a dziesięcioma sekundami, na wyświetlaczu pojawia się komunikat i rozpoczyna się pomiar czasu reakcji. Kiedy użytkownik ponownie naciśnie joystick, uzyskany wynik w milisekundach jest wyświetlany na ekranie i ewentualnie aktualizowany jest rekordowy wynik. Wtedy pojawia się również opcja zresetowania zapisanego w pamięci rekordowego wyniku poprzez przechylenie płytki o kąt 30 stopni w dowolnym kierunku. Pomyślne wykonanie tej operacji zostaje potwierdzone komunikatem na wyświetlaczu. Kolejne rundy rozpoczynamy poprzez przechylenie joysticka w dół, po czym jest powtarzana ta sama sekwencja operacji co w rundzie pierwszej. Po zakończeniu piątej rundy, na ekranie wyświetlany jest komunikat o zakończeniu rozgrywki.

4. Opis działania programu - opis algorytmu

1. Inicjalizacja wszystkich potrzebnych urządzeń peryferyjnych.

2. Rozpoczęcie 5-iteracyjnej pętli.
   (a) Zmierzenie bieżącego oświetlenia i ewentualny negatyw ekranu.
   (b) Wyświetlenie na ekranie komunikatu powitalnego wraz z bieżącym rekordowym wynikiem.
   (c) Oczekiwanie na naciśnięcie joysticka.
   (d) Odliczanie od 3 do 0 w 1-sekundowych odstępach i wyświetlanie na ekranie napisów '3', '2', '1' oraz 'Czekaj...' co każdą sekundę wraz z akompaniującym dźwiękiem.
   (e) Generacja pseudolosowej wartości między 1000 a 10000 na podstawie ziarna w postaci wartości bieżącego oświetlenia otoczenia pozyskanej z czujnika światła i rozpoczęcie oczekiwania o czasie trwania równym uzyskanej wartości.
   (f) Rozpoczęcie pomiaru czasu reakcji wraz z wyświetleniem na ekranie komunikatu 'Start!' i jego przerwanie w momencie naciśnięcia joysticka.
   (g) Wyświetlenie wartości zmierzonego czasu w milisekundach na ekranie.
   (h) Porównanie wartości uzyskanego wyniku z wartością rekordową przechowywaną w module EEPROM i jego aktualizacja w przypadku, gdy uzyskany wynik posiada niższą wartość.
   (i) Oczekiwanie na przechylenie joysticka w dół, w międzyczasie - sprawdzanie za pomocą akcelerometru czy nie nastąpiło przechylenie płytki (jeżeli nastąpiło, wartość rekordowego wyniku przechowywana w module EEPROM ustawiana jest na 0 oraz na ekranie jest wyświetlany komunikat 'Reset HS').
3. Przejście do kolejnej iteracji i powtórzenie operacji.
4. Zmierzenie bieżącego oświetlenia i ewentualny negatyw ekranu.
5. Wyświetlenie na ekranie komunikatu o zakończeniu gry.

5. Opis funkcjonalności poszczególnych elementów

1. Timer

Na płytce znajdują się dwa 32-bitowe czasomierze (ozn. timer0 i timer1). Oba z nich zostały wykorzystane w programie. Czasomierze działają na zasadzie zliczania cykli zegara peryferyjnego (PCLK), zapewnianego przez zegar systemowy.

W programie wykorzystane są następujące rejestry timerów:

TCR (Timer Control Register):

- Adres: 0x004.
- 32-bitowy rejestr kontrolujący stan rejestrów TC i PC.
- Zawiera bity sterujące służące do:
  - Uruchamiania/zatrzymywania rejestrów TC i PC (bit 0 – Counter Enable). Kiedy wartość bitu jest równa 1, rejestry są aktywne. W przeciwnym wypadku, zostają zdezaktywowane.

– Resetowania rejestrów TC oraz PC (bit 1 – Counter Reset). Kiedy wartość bitu jest równa 1, w kolejnym cyklu zegara PCLK następuje reset rejestrów. Rejestry pozostają zresetowane do czasu ustawienia wartości bitu na 0.

TC (Timer Counter):

- Adres: 0x008
- 32-bitowy rejestr inkrementowany co PR+1 cykli zegara peryferyjnego (PCLK).
- Jest kontrolowany przez rejestr TCR.
- Wartość w TC odzwierciedla liczbę przepełnień rejestru PC timera od momentu jego uruchomienia.

PR (Prescale Register):

- Adres: 0x00C.
- 32-bitowy rejestr definiujący maksymalną wartość rejestru PC.
- Kiedy wartość rejestru PC osiągnie wartość równą tej w rejestrze PR, następuje inkrementacja rejestru TC i wyczyszczenie rejestru PC.

Uwaga: Zegar systemowy pracuje z częstotliwością 72 MHz. Poniżej znajduje się fragment kodu programu ustawiający tę wartość:

```
#define __XTAL (12000000UL) /* Oscillator frequency */
#define __SYS_OSC_CLK (__XTAL) /* Main oscillator frequency */
#define SYSPLICTRL_Val 0x00000025
LPC_SYSCON->SYSPLLCTRL = SYSPLLCTRL_Val;
SystemCoreClock = __SYS_OSC_CLK * ((LPC_SYSCON->SYSPLLCTRL & 0x01F) + 1);
/* 12MHz * (5+1) = 72MHz */
```

$$12\,\text{MHz} \times (0x25 \& 0x1F + 1) = 12\,\text{MHz} \times (5 + 1) = 72\,\text{MHz}$$

PC (Prescale Counter):

- Adres: 0x010.
- 32-bitowy rejestr zliczający.
- Wartość rejestru jest inkrementowana z każdym cyklem zegara PCLK.
- Kiedy wartość rejestru PC osiągnie wartość równą tej w rejestrze PR, w kolejnym cyklu zegara PCLK następuje inkrementacja rejestru TC i wyczyszczenie rejestru PC.

MCR (Match Control Register):

- Adres: 0x014.
- 32-bitowy rejestr sterujący działaniem rejestrów MR.
- Umożliwia ustalenie operacji wykonywanej, gdy nastąpi dopasowanie rejestrów TC i rejestrów MR0-3.
- Bity sterujące 0-11 służą do doboru operacji podczas dopasowania rejestru TC z odpowiednim rejestrem MR.

IR (Interrupt Register):

- Adres: 0x000.
- Służy do identyfikacji źródeł wygenerowanych przerwań.

Rozpoczęcie odliczania na czasomierzu i odczytanie wartości bieżącego upływu czasu:

```
1 init_timer32(0, 72000);
2 LPC_TMR32B0->TCR = 0x02;
3 uint32_t prescalerValue = ((SystemFrequency/LPC_SYSCON->SYSAHBCLKDIV) /
      1000) - 1;
4 LPC_TMR32B0->PR = prescalerValue;
5 LPC_TMR32B0->MCR = 0x00;
6 LPC_TMR32B0->TCR = 0x01;
7 waitForJoystickCenterClick();
8 uint32_t reactionTimeMs = LPC_TMR32B0->TC;
```

Opis wartości w rejestrach:

- Wartość w rejestrze TCR: 0x02 – reset i zatrzymanie liczników rejestrów TC oraz PC.
- Wartość w rejestrze PR: częstotliwość zegara systemowego (SystemFrequency/SystemCoreClock) podzielona przez 1000 (by uzyskiwać wynik w milisekundach), a następnie dekrementowana o 1 (ponieważ rejestr TC jest inkrementowany co PR+1 cykli PCLK).
- Wartość w rejestrze MCR: 0x00 – reset rejestru / brak czynności przy dopasowaniu rejestrów MR i rejestru TC.
- Wartość w rejestrze TCR: 0x01 – uruchomienie liczników rejestrów TC oraz PC.

Opis wykorzystywanej w programie funkcji `init_timer32()`:

```
1 (...)
2 LPC_TMR32B0->MR0 = TimerInterval;
3 (...)
4 LPC_TMR32B0->MCR = 3;
```

Gdzie `TimerInterval` = żądany czas, po którym zostanie zgłoszone przerwanie.

Opis wartości w rejestrach:

- Wartość w rejestrze MR0: żądany czas, po którym zostanie zgłoszone przerwanie.

- Wartość w rejestrze MCR: 3 – generowanie przerwania i reset licznika rejestru TC po dopasowaniu rejestrów TC oraz MR0.

Opis wykorzystywanej w programie funkcji `delay32Ms()`:

```
1         LPC_TMR32B0->TCR = 0x02; /* reset timer */
2         LPC_TMR32B0->PR = 0x00; /* set prescaler to zero */
3         LPC_TMR32B0->MR0 = delayInMs * ((SystemFrequency/LPC_SYSCON->
              SYSAHBCLKDIV) / 1000);
4         LPC_TMR32B0->IR = 0xff; /* reset all interrupts */
5         LPC_TMR32B0->MCR = 0x04; /* stop timer on match */
6         LPC_TMR32B0->TCR = 0x01; /* start timer */
7         /* wait until delay time has elapsed */
8         while (LPC_TMR32B0->TCR & 0x01);
```

Gdzie `delayInMs` = żądany czas oczekiwania w milisekundach.
Opis wartości w rejestrach:

- Wartość w rejestrze IR: 0xff – zresetowanie wszystkich przerwań, zgłoszonych przez rejestry MR0-3.
- Wartość w rejestrze PR: 0x00 – inkrementacja licznika rejestru TC w każdym cyklu PCLK.
- Wartość w rejestrze MR0: żądany czas oczekiwania w milisekundach wymnożony przez częstotliwość zegara podzieloną przez 1000 (aby uzyskać wynik w milisekundach). Wartość w rejestrze SYSAHBCLKDIV jest równa 1.
- W rejestrze MCR bit 2 jest ustawiany na wartość 1 w celu zatrzymania liczników rejestrów TC oraz PC poprzez ustawienie wartości bitu 0 rejestru TCR na 0 przy dopasowaniu rejestrów TC oraz MR0.

Uwaga: w programie jest używana również funkcja `delay32Us()` działająca w taki sam sposób, z wyjątkiem dzielenia częstotliwości zegara przez 1000000 zamiast 1000, aby uzyskać wynik w mikrosekundach.

2. Dźwięk

Realizacja dźwięku odbywa się poprzez filtr dolnoprzepustowy PWM oraz wzmacniacz analogowy LM4811.

Filtr dolnoprzepustowy PWM (PWM Low Pass Filter to Analog Signal):

- Sygnał wejściowy jest przekazywany poprzez pin 2 na porcie GPIO1 (P1.2). Sygnał jest w formie sygnału PWM, co wymaga, aby w celu jego wytworzenia, naprzemiennie ustawiać pin P1.2 w stan wysoki i niski, tworząc w ten sposób falę prostokątną.

Inicjalizacja pinu wejściowego filtra i ustawienie trybu PIO1_2:

```
1 GPIOSetDir( PORT1, 2, 1 );
2 LPC_IOCON->JTAG_nTRST_PIO1_2 = (LPC_IOCON->JTAG_nTRST_PIO1_2 & ~0x7) | 0
      x01;
```

Pin jest ustawiany w stan wysoki i niski za pomocą następujących makr:

```
1 #define P1_2_HIGH() (LPC_GPIO1->DATA |= ((uint16_t)0x1<<2))
2 #define P1_2_LOW() (LPC_GPIO1->DATA &= ~((uint16_t)0x1<<2))
```

Sygnał PWM jest wytwarzany w następujący sposób:

```
P1_2_HIGH();
delay32Us(0, note / (uint32_t)2);
P1_2_LOW();
delay32Us(0, note / (uint32_t)2);
```

Gdzie `note` to żądany okres drgań dźwięku.

- Wejściowy sygnał PWM jest konwertowany na wyjściowy sygnał analogowy.
- Sygnał wyjściowy jest przekazywany do wzmacniacza analogowego LM4811.

Wzmacniacz analogowy LM4811 (LM4811 Analog Amplifier):

- Wejściowy sygnał analogowy jest wzmacniany i przekazywany do głośnika, który emituje falę dźwiękową.

Inicjalizacja pinów wzmacniacza:

```
GPIOSetDir( PORT3, 0, 1 );
GPIOSetDir( PORT3, 1, 1 );
GPIOSetDir( PORT3, 2, 1 );
GPIOSetValue( PORT3, 0, 0 ); //LM4811-clk
GPIOSetValue( PORT3, 1, 0 ); //LM4811-up/dn
GPIOSetValue( PORT3, 2, 0 ); //LM4811-shutdn
```

3. Joystick GPIO

Joystick jest urządzeniem peryferyjnym sterowanym za pomocą GPIO (General Purpose Input/Output). Pozycja joysticka jest odczytywana poprzez sprawdzenie stanu pinów GPIO, do których podłączone są przyciski i osie joysticka. Mikrokontroler skanuje piny i wykrywa zmiany stanu, które oznaczają ruch joysticka.

Konfiguracja pinów GPIO

Joystick znajduje się na porcie drugim mikrokontrolera i ma 4 piny, które w funkcji `joystick_init` mają wartość ustawioną na 0, co oznacza, że ich kierunek jest ustawiony na "wejście"(jeśli byłoby 1, oznaczałoby kierunek "wyjście"):

```
GPIOSetDir( PORT2, 0, 0 );
GPIOSetDir( PORT2, 1, 0 );
GPIOSetDir( PORT2, 2, 0 );
GPIOSetDir( PORT2, 3, 0 );
```

Odczyt stanu joysticka

Joystick korzysta z flag bitowych, które pozwalają na reprezentowanie wielu kierunków jednocześnie za pomocą operacji bitowych OR. Na przykład, jeśli zarówno kierunek góra, jak i prawo są naciśnięte, wartość łączna będzie `JOYSTICK_UP | JOYSTICK_RIGHT`, co daje `0x02 | 0x10 = 0x12`.

Podsumowanie

Dzięki takiemu podejściu możliwe jest rozpoznanie wielu stanów joysticka oraz ich kombinacji, co pozwala na intuicyjną obsługę urządzenia w programie.

4. Interfejs SPI

SPI (Serial Peripheral Interface) to szybki, synchroniczny interfejs szeregowy wykorzystywany do komunikacji pomiędzy mikrokontrolerami a różnego rodzaju układami peryferyjnymi, takimi jak wyświetlacze OLED, czujniki czy pamięci. Architektura SPI oparta jest na modelu master-slave, gdzie jeden układ (master) steruje transmisją, a pozostałe (slave) odpowiadają na żądania.

Podstawowe linie SPI

Typowa magistrala SPI składa się z czterech linii:

- **SCK (Serial Clock)** – linia zegara generowana przez mastera, synchronizująca transmisję.
- **MOSI (Master Out, Slave In)** – linia danych od mastera do slave'a.
- **MISO (Master In, Slave Out)** – linia danych od slave'a do mastera.
- **SS/CS (Slave Select/Chip Select)** – linia wyboru układu slave, aktywna w stanie niskim.

Zasada działania

Transmisja SPI jest pełnodupleksowa – dane mogą być przesyłane w obu kierunkach jednocześnie. Master inicjuje komunikację, generując sygnał zegarowy i aktywując wybranego slave'a przez obniżenie linii SS. W każdym cyklu zegara jeden bit jest przesyłany z mastera do slave'a (MOSI) i jednocześnie jeden bit z slave'a do mastera (MISO).

Tryby pracy (CPOL/CPHA)

SPI posiada cztery tryby pracy, określane przez polaryzację (CPOL) i fazę (CPHA) sygnału zegara:

- **Mode 0:** CPOL=0, CPHA=0 – dane próbkowane na narastającym zboczu zegara.
- **Mode 1:** CPOL=0, CPHA=1 – dane próbkowane na opadającym zboczu zegara.
- **Mode 2:** CPOL=1, CPHA=1 – dane próbkowane na opadającym zboczu zegara, zegar w stanie spoczynku wysoki.
- **Mode 3:** CPOL=1, CPHA=0 – dane próbkowane na narastającym zboczu zegara, zegar w stanie spoczynku wysoki.

Wybór odpowiedniego trybu zależy od wymagań urządzenia peryferyjnego. Niezgodność trybów po obu stronach uniemożliwi poprawną komunikację.

Konfiguracja SPI w LPC1343

W programie inicjalizacja interfejsu SPI wygląda następująco:

```
1 LPC_SYSCON->PRESETCTRL |= (0x1<<0);          // Wlaczenie zasilania
    peryferii SPI
2 LPC_SYSCON->SYSAHBCLKCTRL |= (1<<11);        // Taktowanie peryferii SPI
3 LPC_SYSCON->SSPCLKDIV = 0x02;                // Ustawienie dzielnika
    zegara SPI
4
5 LPC_IOCON->PIO0_8 &= ~0x07;
6 LPC_IOCON->PIO0_8 |= 0x01;                   // SSP MISO
7 LPC_IOCON->PIO0_9 &= ~0x07;
8 LPC_IOCON->PIO0_9 |= 0x01;                   // SSP MOSI
```

Pierwsze trzy linie włączają zasilanie i taktowanie oraz ustawiają dzielnik zegara dla peryferium SPI. Kolejne linie konfigurowane są piny mikrokontrolera do funkcji MISO i MOSI.

Uwaga praktyczna: W przypadku kilku urządzeń slave, każde z nich powinno mieć osobną linię SS, sterowaną przez mastera. W stanie nieaktywnym linie SS należy podciągnąć do stanu wysokiego za pomocą rezystorów, aby uniknąć przypadkowej aktywacji urządzeń i konfliktów na magistrali.

Topologie połączeń

- **Jeden master, jeden slave:** najprostsza konfiguracja, wszystkie linie są połączone bezpośrednio.
- **Jeden master, wiele slave:** master wybiera aktywnego slave'a przez odpowiednią linię SS. Pozostałe urządzenia pozostają nieaktywne.
- **Daisy-chain:** slave'y połączone szeregowo, dane przesyłane przez kolejne urządzenia, jeden wspólny sygnał SS.

Zalety i ograniczenia SPI

Zalety:

- Bardzo wysoka szybkość transmisji (do kilkudziesięciu Mb/s).
- Prosta implementacja sprzętowa i programowa.
- Pełny dupleks – jednoczesna transmisja i odbiór danych.

Ograniczenia:

- Brak standaryzacji protokołu warstwy wyższej (każde urządzenie może mieć własny sposób interpretacji danych).
- Konieczność osobnej linii SS dla każdego slave'a w klasycznej topologii.
- Brak możliwości adresowania urządzeń na magistrali (w przeciwieństwie do I2C).

Zastosowanie w projekcie

W projekcie interfejs SPI został wykorzystany do komunikacji z wyświetlaczem OLED. Dzięki wysokiej przepustowości możliwa jest szybka aktualizacja zawartości ekranu, co jest kluczowe w grach wymagających dynamicznego odświeżania grafiki.

Przykład inicjalizacji pinów dla SPI:

```
GPIOSetDir(PORT0, 0, 1 );
GPIOSetDir(PORT1, 10, 1 );
GPIOSetDir(PORT2, 7, 1 );
GPIOSetDir(PORT0, 2, 1 );
```

Powyższy kod ustawia odpowiednie piny jako wyjścia, co pozwala na poprawne sterowanie wyświetlaczem OLED przez magistralę SPI.

Podsumowanie: SPI jest jednym z najczęściej wykorzystywanych interfejsów do komunikacji z szybkimi peryferiami. Warto pamiętać o poprawnej konfiguracji linii SS oraz zgodności trybów pracy, aby zapewnić bezproblemową transmisję danych.

5. Wyświetlacz OLED

Wyświetlacz OLED użyty w projekcie ma rozdzielczość 96 x 64 piksele i pracuje w oparciu o interfejs SPI. Dzięki technologii OLED zapewnia głęboki kontrast oraz wyraziste kolory, umożliwiając prezentację czytelnych informacji na niewielkiej powierzchni ekranu. W projekcie zastosowano funkcję `oled_putChar`, która zamienia podany znak na odpowiednie piksele wyświetlane na ekranie, korzystając z tablicy `font5x7` zawierającej bitmapy znaków.

Budowa i podłączenie wyświetlacza

Wyświetlacz OLED o przekątnej 0,95"i rozdzielczości 96x64 piksele korzysta z palety barw RGB i jest wyposażony w kontroler SSD1331. Moduł komunikuje się poprzez interfejs SPI, a napięcie zasilania wynosi od 3,3 do 5 V. Dzięki kompaktowym wymiarom (ok. 27 x 31 mm) oraz wlutowanym goldpinom, integracja z projektem jest bardzo prosta[2][6].

Wyprowadzenia modułu:

- GND – masa układu
- VDD – napięcie zasilania (3,3–5 V)
- SCK – linia zegarowa SPI
- SDA – linia danych SPI (MOSI)
- RES – reset
- DC – dane/komenda
- CS – wybór układu (chip select)

Główne komponenty wyświetlacza OLED

- **Interfejs SPI** – umożliwia szybką komunikację z mikrokontrolerem oraz transfer komend i danych obrazu.
- **Kontroler SSD1331** – odpowiada za interpretację komend, zarządzanie pamięcią RAM wyświetlacza oraz generowanie sygnałów sterujących pikselami.
- **Pamięć RAM** – przechowuje dane obrazu, które mają być wyświetlane. Kontroler cyklicznie odczytuje te dane podczas odświeżania ekranu (typowa częstotliwość odświeżania to ok. 140 Hz).
- **Układ sterujący** – steruje jasnością i kolorem diod OLED na podstawie danych z RAM.
- **Panel OLED** – matryca diod generujących obraz.

Przepływ danych i obsługa wyświetlacza

1. **Przesyłanie komend** – mikrokontroler wysyła przez SPI komendy sterujące (np. ustawienia parametrów wyświetlania, reset, inicjalizacja).
2. **Transfer danych obrazu** – mikrokontroler przesyła dane obrazu (bitmapy, znaki) do pamięci RAM wyświetlacza.
3. **Cykl odświeżania** – kontroler SSD1331 pobiera dane z RAM i steruje diodami OLED, generując widoczny obraz.

Przykład kodu i inicjalizacji pinów

W kodzie projektu wyświetlacz obsługiwany jest przez funkcje, które zamieniają znaki na bitmapy i przesyłają je przez SPI:

```
/* Przyklad fragmentu fontu */
const uint8_t font5x7[][5] = {
    /* '?' */
    { 0x22, 0x01, 0x51, 0x09, 0x06 },
    /* kolejne znaki... */
};
```

Inicjalizacja pinów do obsługi wyświetlacza przez SPI:

```
GPIOSetDir(PORT0, 0, 1 );
GPIOSetDir(PORT1, 10, 1 );
GPIOSetDir(PORT2, 7, 1 );
GPIOSetDir(PORT0, 2, 1 );
```

Ustawienie pinu zasilania na niski stan (wyłączenie wyświetlacza):

```
GPIOSetValue(PORT1, 10, 0 );
```

Po inicjalizacji parametrów, włączenie wyświetlacza:

```
GPIOSetValue(PORT1, 10, 1 );
```

Cechy i zalety wyświetlacza OLED w projekcie

- **Wysoka rozdzielczość** – 96x64 px pozwala na prezentację szczegółowych informacji i prostych grafik.
- **Nieskończony kontrast** – technologia OLED zapewnia idealną czerń i wysoką czytelność nawet przy niewielkich rozmiarach.
- **Bogata paleta barw** – do 65536 kolorów (16 bitów na piksel).
- **Szybka komunikacja SPI** – umożliwia dynamiczne odświeżanie ekranu, niezbędne w aplikacjach interaktywnych.
- **Elastyczne zasilanie i łatwy montaż** – szeroki zakres napięć oraz wlutowane goldpiny.

Podsumowanie: Wyświetlacz OLED jest kluczowym elementem interfejsu użytkownika w projekcie – umożliwia prezentację wyników, komunikatów oraz dynamicznych zmian stanu gry w sposób czytelny i atrakcyjny wizualnie.

6. Interfejs I2C

I2C (Inter-Integrated Circuit) to synchroniczny, szeregowy protokół komunikacyjny typu master-slave, wykorzystywany do łączenia mikrokontrolerów z układami peryferyjnymi, takimi jak czujniki, pamięci czy ekspandery portów. W projekcie interfejs I2C służy do komunikacji m.in. z czujnikiem oświetlenia ISL29003.

Zasada działania magistrali I2C

Magistrala I2C wykorzystuje dwie linie:

- **SDA** (Serial Data Line) – linia danych,
- **SCL** (Serial Clock Line) – linia zegara.

Obie linie są typu otwarty kolektor, wymagają podciągnięcia do napięcia zasilania przez rezystory (najczęściej 4,7 kΩ), co pozwala na podłączanie wielu urządzeń do wspólnej magistrali.

Transmisja danych odbywa się w postaci komunikatów (messages) podzielonych na ramki (frames):

- **Start Condition** – sygnalizowana przez przejście SDA z wysokiego na niski poziom przy wysokim SCL.
- **Address Frame** – 7- lub 10-bitowy adres slave'a oraz bit R/W (0 – zapis, 1 – odczyt).
- **Data Frames** – każda ramka to 8 bitów danych, przesyłanych z najbardziej znaczącym bitem (MSB) jako pierwszym.
- **ACK/NACK** – po każdej ramce odbiorca potwierdza odbiór przez wystawienie bitu ACK (0) lub NACK (1).
- **Stop Condition** – przejście SDA z niskiego na wysoki poziom przy wysokim SCL.

Adresowanie pozwala na podłączenie do 128 urządzeń slave (przy adresacji 7-bitowej) bez konieczności stosowania dodatkowych linii wyboru, jak w SPI.

Konfiguracja sprzętowa na LPC1343

Aby uruchomić interfejs I2C w trybie master:

```
LPC_SYSCON->PRESETCTRL |= (0x1<<1);          // Reset I2C
LPC_SYSCON->SYSAHBCLKCTRL |= (1<<5);         // Wlaczenie zegara I2C

LPC_IOCON->PIO0_4 &= ~0x3F;
LPC_IOCON->PIO0_4 |= 0x01;                   // SCL
LPC_IOCON->PIO0_5 &= ~0x3F;
LPC_IOCON->PIO0_5 |= 0x01;                   // SDA

LPC_I2C->SCLL = 0x180;                       // Czas trwania stanu niskiego
    SCL
LPC_I2C->SCLH = 0x180;                       // Czas trwania stanu
   wysokiego SCL
```

Częstotliwość sygnału zegarowego SCL wyznacza wzór:

$$f_{I2C} = \frac{PCLK}{2 \times (SCLH + SCLL)}$$

Dla PCLK = 72 MHz oraz SCLH = SCLL = 384, uzyskujemy $f_{I2C} \approx 46,9$ kHz.

Rejestry I2C w LPC1343

- **I2CONSET/I2CONCLR** – rejestry ustawień i kasowania bitów kontrolnych.
- **I2STAT** – rejestr statusu (kody stanu transmisji).
- **I2DAT** – rejestr danych (do wysyłania i odbioru bajtów).
- **I2SCLH/I2SCLL** – rejestry ustawiające długość impulsu zegara SCL.
- **I2ADR0-3** – rejestry adresowe dla trybu slave.
- **I2DATA BUFFER** – bufor odbiorczy.

Protokół komunikacji z ISL29003

Czujnik światła ISL29003 komunikuje się przez I2C jako slave (domyślny adres: 0x44). Odczyt danych z czujnika wymaga wysłania adresu rejestru, z którego chcemy czytać (np. adres LSB lub MSB wyniku pomiaru), a następnie odczytu bajtu danych.

Przykładowa funkcja odczytu natężenia światła:

```
uint32_t light_read(void) {
    uint32_t data = 0;
    uint8_t buf[1];
    buf[0] = ADDR_LSB_SENSOR;
    I2CWrite(LIGHT_I2C_ADDR, buf, 1);
    I2CRead(LIGHT_I2C_ADDR, buf, 1);
    data = buf[0];
    buf[0] = ADDR_MSB_SENSOR;
    I2CWrite(LIGHT_I2C_ADDR, buf, 1);
    I2CRead(LIGHT_I2C_ADDR, buf, 1);
    data = (buf[0] << 8) | data;
    return (range * data / width);
}
```

Opis działania:

- Wysłanie adresu rejestru LSB, odczyt młodszego bajtu.
- Wysłanie adresu rejestru MSB, odczyt starszego bajtu.
- Połączenie bajtów do postaci 16-bitowej liczby.
- Przeliczenie wartości na jednostki luksów zgodnie z wybranym zakresem i rozdzielczością czujnika (parametry `range` i `width`).

Charakterystyka ISL29003

ISL29003 to zintegrowany czujnik światła z 16-bitowym przetwornikiem ADC, umożliwiający pomiar natężenia światła w szerokim zakresie (do 64 000 luksów). Czujnik pozwala na wybór zakresu pomiarowego, ustawienie czasu integracji oraz generuje dane proporcjonalne do natężenia światła. Komunikacja przez I2C umożliwia łatwą integrację z mikrokontrolerem i odczyt danych w postaci cyfrowej[2][3][5][6].

Podsumowanie: Interfejs I2C zapewnia prostą i niezawodną komunikację z wieloma układami peryferyjnymi na wspólnej magistrali, a dzięki adresowaniu umożliwia obsługę wielu urządzeń bez konieczności dodatkowych linii wyboru. Implementacja obsługi czujnika ISL29003 przez I2C pozwala na szybki i precyzyjny odczyt natężenia światła w czasie rzeczywistym.

7. Linijka diod

Linijka diod LED w projekcie jest realizowana za pomocą układu PCA9532, będącego 16-bitowym sterownikiem LED z interfejsem I2C. Umożliwia on indywidualne sterowanie stanem oraz jasnością do 16 diod LED, a także generowanie efektów takich jak miganie czy płynne ściemnianie.

Charakterystyka PCA9532

PCA9532 to specjalizowany ekspander I/O zoptymalizowany do sterowania diodami LED. Najważniejsze cechy:

- 16 wyjść typu open-drain, mogących bezpośrednio sterować diodami LED (do 25 mA na wyjście, 200 mA na układ).
- Możliwość ustawienia każdego wyjścia w jednym z czterech trybów: wyłączony (OFF), włączony (ON), miganie z częstotliwością i współczynnikiem wypełnienia PWM0, miganie z PWM1.
- Dwa niezależnie programowalne generatory PWM (częstotliwość od 0,591 Hz do 152 Hz, 256 poziomów jasności).
- Obsługa I2C do 400 kHz, zgodność z SMBus.
- Możliwość pracy kilku układów na jednej magistrali (3 piny adresowe A0-A2).
- Po resecie wszystkie diody są wyłączone (stan wysoki na wyjściach).

Podłączenie i adresacja

Wyjścia LED0–LED15 są bezpośrednio połączone z diodami LED przez rezystory ograniczające prąd. Linie SDA i SCL są podciągnięte do zasilania przez rezystory 10 kΩ. Adres układu na magistrali I2C ustala się przez odpowiednie podłączenie pinów A0–A2 do masy lub zasilania (w projekcie: 0x60).

Sterowanie diodami w programie

W programie do sterowania diodami wykorzystywane są maski bitowe:

- `ledOffMask` – bity ustawione na 1 oznaczają diody wyłączone,
- `ledOnMask` – bity ustawione na 1 oznaczają diody włączone.

W każdej rundzie gry maska diod włączonych jest przesuwana w lewo, aby wizualizować postęp (np. `0x0E`, `0x3C`, `0xF8` itd.).

Najpierw zerowane są wszystkie bity zmiennej `ledStateShadow` przez operację:

```
ledStateShadow &= (ledOffMask & 0xFFFF);
```

Następnie ustawiane są bity odpowiadające diodom do włączenia:

```
ledStateShadow |= ledOnMask;
```

Po ustaleniu stanu wywoływana jest funkcja `setLeds()`, która przekazuje stan do PCA9532 przez I2C.

Przykład ustawiania stanów diod

Tablica `ls[]` przechowuje stany LED w 4 bajtach (po 4 diody na bajt). Funkcja `setLsStates()` ustawia odpowiednie bity:

```
#define IS_LED_SET(bit, x) ( ((x) & (bit)) ? 1 : 0 )
for (int i = 0; i < 4; i++) {
    ls[i] |= (
        (IS_LED_SET(LED4, states) * mode << 0) |
        (IS_LED_SET(LED5, states) * mode << 2) |
        (IS_LED_SET(LED6, states) * mode << 4) |
        (IS_LED_SET(LED7, states) * mode << 6)
    );
    states >>= 4;
}
```

Makro `IS_LED_SET` sprawdza, czy dany bit odpowiadający diodzie jest ustawiony w masce stanu.

Definicje bitów LED

Każda dioda ma przypisany bit:

```
#define LED4  0x0001 // LED0 na plytce
#define LED5  0x0002 // LED1
// ...
#define LED19 0x8000 // LED15
```

Uwagi praktyczne

- PCA9532 posiada wewnętrzny oscylator – nie wymaga dodatkowych elementów zewnętrznych.
- Wyjścia typu open-drain umożliwiają bezpieczne sterowanie diodami LED z różnych napięć (zasilanie układu 2,3–5,5 V).
- Diody LED można ściemniać i migać programowo, co pozwala na efekty wizualne bez obciążania mikrokontrolera.
- Nieużywane wyjścia mogą być wykorzystane jako GPIO.

Podsumowanie: Zastosowanie PCA9532 umożliwia elastyczne i wydajne sterowanie linijką diod LED przez interfejs I2C, z możliwością programowej kontroli jasności, efektów migania oraz łatwej rozbudowy o kolejne linie LED bez zajmowania dodatkowych pinów mikrokontrolera.

8. Moduł EEPROM

Do zapisu najwyższego wyniku gry wykorzystujemy pamięć nieulotną typu EEPROM, model 24LC08B, podłączoną do magistrali I2C pod adresami od 0x50 do 0x53. Pamięć ta posiada 1024 bajty podzielone na cztery bloki po 256 bajtów każdy. Operacje zapisu i odczytu realizowane są poprzez interfejs I2C, który należy uprzednio zainicjalizować.

Adresowanie i organizacja pamięci

Adres urządzenia EEPROM 24LC08B składa się z:

- Stałej części adresu (bity b7-b4) ustawionej na 1010,
- Trzech bitów A2, B1, B0 określających blok pamięci i konfigurację pinów adresowych,
- Bitu R/W określającego operację odczytu (1) lub zapisu (0).

Dzięki temu możliwe jest kaskadowanie do czterech bloków pamięci (4 x 256 bajtów), co daje łącznie 1024 bajty.

Funkcje zapisu i odczytu

Do zapisu danych wykorzystywana jest funkcja:

```
1 void eeprom_write(uint8_t* buf, uint16_t offset, uint16_t len);
```

gdzie:

- `buf` – wskaźnik na dane do zapisania,
- `offset` – offset (adres początkowy) względem początku pamięci EEPROM,
- `len` – liczba bajtów do zapisania.

Do odczytu danych służy funkcja:

```
void eeprom_read(uint8_t* buf, uint16_t offset, uint16_t len);
```

gdzie:

- `buf` – wskaźnik na bufor, do którego zostaną zapisane odczytane dane,
- `offset` – offset (adres początkowy) względem początku pamięci EEPROM,
- `len` – liczba bajtów do odczytania.

Procedura odczytu

Operacja odczytu wymaga najpierw wysłania adresu wewnętrznego pamięci, z którego chcemy czytać, poprzez zapis na magistrali I2C, a następnie odczytu danych. Przykładowy fragment kodu ilustrujący tę procedurę:

```
uint8_t addr = EEPROM_I2C_ADDR1 + (offset / EEPROM_BLOCK_SIZE);
uint8_t off = offset % EEPROM_BLOCK_SIZE;
I2CWrite((addr << 1), (uint8_t*)&off, 1);
for (int i = 0; i < 0x2000; i++); // opoznienie na zakonczenie operacji
I2CRead((addr << 1), buf, len);
```

- `addr` – adres urządzenia EEPROM na magistrali I2C, zależny od bloku pamięci,
- `off` – offset wewnątrz bloku pamięci,
- `I2CWrite` – funkcja wysyłająca adres odczytu,
- `I2CRead` – funkcja odczytująca dane z pamięci EEPROM.

Opis działania

- Po wysłaniu adresu wewnętrznego następuje opóźnienie (np. pętla `for`), które pozwala na zakończenie wewnętrznego cyklu zapisu lub przygotowanie danych do odczytu.
- Następnie następuje odczyt żądanej liczby bajtów z pamięci EEPROM.
- Podczas zapisu EEPROM 24LC08B automatycznie wykonuje wewnętrzny cykl zapisu, który trwa do kilku milisekund. W tym czasie urządzenie nie odpowiada na kolejne żądania na magistrali I2C.
- Możliwe jest wykonywanie operacji page write (zapis do 16 bajtów na raz), gdzie adres wewnętrzny automatycznie się inkrementuje, a po przekroczeniu rozmiaru strony następuje nadpisanie danych od początku strony.

Zalecenia i uwagi

- Pin WP (Write Protect) powinien być podłączony do masy (VSS), aby umożliwić zapis do całej pamięci.
- Należy uwzględnić czas wewnętrznego cyklu zapisu EEPROM (maksymalnie 5 ms) i nie inicjować kolejnych operacji zapisu przed jego zakończeniem.

- W przypadku odczytu sekwencyjnego można czytać kolejne bajty bez ponownego wysyłania adresu, aż do końca pamięci, po czym adresowanie „zawija się" do początku.
- Podczas programowania strony (page write) należy uważać, aby nie przekroczyć 16 bajtów, gdyż nadmiar danych spowoduje nadpisanie początkowych bajtów strony.

Podsumowanie

Moduł EEPROM 24LC08B zapewnia trwałe przechowywanie danych, takich jak rekordowe wyniki gry, dzięki czemu możliwe jest ich zachowanie po wyłączeniu zasilania. Komunikacja przez interfejs I2C oraz mechanizmy adresowania blokowego umożliwiają efektywne zarządzanie pamięcią i prostą integrację z mikrokontrolerem LPC1343.

9. Akcelerometr

Akcelerometr MMA7455L to trójosiowy cyfrowy czujnik MEMS, umożliwiający pomiar przyspieszenia wzdłuż osi X, Y i Z. W projekcie wykorzystano komunikację przez interfejs I2C. Urządzenie to znajduje szerokie zastosowanie w elektronice użytkowej, m.in. do detekcji ruchu, orientacji, wstrząsów czy swobodnego opadania.

Główne cechy MMA7455L

- Cyfrowe wyjście (I2C lub SPI, w projekcie użyto I2C)
- Pomiar przyspieszenia w trzech osiach (X, Y, Z)
- Zakresy pomiarowe: $\pm 2\,g$, $\pm 4\,g$, $\pm 8\,g$ (w projekcie: $\pm 2\,g$)
- Czułość: 64 LSB/g przy 2g (tryb 10-bitowy)
- Możliwość generowania dwóch niezależnych przerwań (wykrywanie wstrząsów, swobodnego opadania, wibracji)
- Napięcie zasilania: 2,5–5,5 V (dzięki wbudowanemu LDO i układom poziomującym)
- Niska konsumpcja prądu: ok. 0,5 mA w trybie pracy, 26–42 µA w standby[3]

Zasada działania i podłączenie

Układ MMA7455L to mikromechaniczny system MEMS – miniaturowa masa zawieszona na sprężynach, której odchylenie pod wpływem przyspieszenia jest mierzone i przetwarzane na sygnał cyfrowy przez wbudowany przetwornik ADC. Moduł posiada wyprowadzenia do komunikacji I2C (SDA, SCL) oraz dwa wyjścia przerwań (INT1, INT2), które można podłączyć do mikrokontrolera. W celu ochrony linii przerwań i komunikacyjnych stosuje się rezystory szeregowe 270 Ω.

Odczyt danych z akcelerometru

Do odczytu przyspieszenia wzdłuż trzech osi wykorzystywana jest funkcja `acc_read()`:

```
void acc_read (int8_t *x, int8_t *y, int8_t *z)
{
    uint8_t buf[1];
    while ((getStatus() & ACC_STATUS_DRDY) == 0);
    buf[0] = ACC_ADDR_XOUT8;
    I2CWrite(ACC_I2C_ADDR, buf, 1);
    I2CRead(ACC_I2C_ADDR, buf, 1);
    *x = (int8_t)buf[0];

    buf[0] = ACC_ADDR_YOUT8;
    I2CWrite(ACC_I2C_ADDR, buf, 1);
    I2CRead(ACC_I2C_ADDR, buf, 1);
    *y = (int8_t)buf[0];

    buf[0] = ACC_ADDR_ZOUT8;
    I2CWrite(ACC_I2C_ADDR, buf, 1);
    I2CRead(ACC_I2C_ADDR, buf, 1);
    *z = (int8_t)buf[0];
}
```

Opis działania:

- Inicjalizowany jest jednobajtowy bufor do komunikacji I2C.
- Program oczekuje na ustawienie flagi gotowości danych (`ACC_STATUS_DRDY`) przez funkcję `getStatus()`.
- Następnie dla każdej osi (X, Y, Z):
  - Do bufora wpisywany jest adres rejestru danej osi (`ACC_ADDR_XOUT8`, `YOUT8`, `ZOUT8`).
  - Wysyłane jest żądanie odczytu przez I2C.
  - Odczytana wartość konwertowana jest na `int8_t` i zapisywana pod wskazany adres.

Inicjalizacja akcelerometru

Funkcja `acc_init()` ustawia tryb pracy na pomiarowy i wybiera zakres pomiarowy:

```
void acc_init (void)
{
    setModeControl( (ACC_MCTL_MODE (ACC_MODE_MEASURE)
        | ACC_MCTL_GLVL(ACC_RANGE_2G) ));
}
```

Opis:

- `ACC_MCTL_MODE(ACC_MODE_MEASURE)` – ustawia tryb pracy na pomiarowy (measurement mode).
- `ACC_MCTL_GLVL(ACC_RANGE_2G)` – ustawia zakres pomiarowy na $\pm 2g$, co jest typowe dla aplikacji użytkowych.
- Parametry są przekazywane do rejestru kontrolnego przez funkcję `setModeControl()`.

Przetwarzanie i interpretacja danych

Odczytane wartości są liczbami ze znakiem (`int8_t`), których zakres odpowiada wybranemu zakresowi pomiarowemu. Przykładowo, dla zakresu $\pm 2g$, wartość 64 odpowiada przyspieszeniu 1g (czułość 64 LSB/g)[3]. Dzięki temu możliwe jest wykrywanie ruchów, przechyłów, wstrząsów czy swobodnego opadania układu.

Uwagi praktyczne

- W projekcie akcelerometr służy m.in. do wykrywania przechylenia płytki (np. do resetowania rekordu).
- Dwa wyjścia przerwań mogą być wykorzystane do szybkiego reagowania na zdarzenia bez konieczności ciągłego odpytywania czujnika.
- Dzięki szerokiemu zakresowi napięć oraz wbudowanym układom poziomującym, moduł jest kompatybilny z większością mikrokontrolerów 3,3 V i 5 V.

Podsumowanie: MMA7455L umożliwia precyzyjny pomiar przyspieszenia w trzech osiach, jest łatwy w integracji dzięki interfejsowi I2C i stanowi uniwersalny komponent do detekcji ruchu, orientacji oraz zdarzeń dynamicznych w systemach wbudowanych.

10. Czujnik oświetlenia ISL29003

ISL29003 to zintegrowany czujnik światła wyposażony w 16-bitowy przetwornik ADC oraz programowalny zakres pomiarowy w luksach. Czujnik obsługuje sprzętowe wyjście przerwania, które pozostaje aktywne w stanie niskim do momentu zresetowania go przez hosta za pośrednictwem interfejsu I2C. ISL29003 został zaprojektowany do pracy z napięciem zasilania od 2,5 V do 3,3 V. Jest stale podłączony do magistrali I2C, a jego wyjście przerwania może być podłączone do pinu PIO2_5 mikrokontrolera. W celu ochrony linii przed uszkodzeniem, w przypadku gdy sygnałem steruje więcej niż jeden sterownik, zastosowano szeregowy rezystor o wartości 270 Ω.

Odczyt natężenia światła – funkcja `light_read()`

Do odczytu natężenia światła wykorzystywana jest funkcja `light_read()`, która komunikuje się z czujnikiem poprzez protokół I2C:

```
uint32_t light_read(void)
{
    uint32_t data = 0;
    uint8_t buf[1];
    buf[0] = ADDR_LSB_SENSOR;
    I2CWrite(LIGHT_I2C_ADDR, buf, 1);
    I2CRead(LIGHT_I2C_ADDR, buf, 1);
    data = buf[0];
    buf[0] = ADDR_MSB_SENSOR;
    I2CWrite(LIGHT_I2C_ADDR, buf, 1);
    I2CRead(LIGHT_I2C_ADDR, buf, 1);
    data = (buf[0] << 8 | data);
    return (range * data / width);
}
```

Opis działania funkcji:

- **Odczyt młodszego bajtu (LSB):**
  - `buf[0] = ADDR_LSB_SENSOR;` – ustawia bufor na adres młodszego bajtu danych czujnika.
  - `I2CWrite(LIGHT_I2C_ADDR, buf, 1);` – wysyła adres rejestru do czujnika przez I2C.
  - `I2CRead(LIGHT_I2C_ADDR, buf, 1);` – odczytuje młodszy bajt danych z czujnika i zapisuje go w `buf[0]`.
  - `data = buf[0];` – zapisuje odczytane dane w zmiennej `data`.
- **Odczyt starszego bajtu (MSB):**
  - `buf[0] = ADDR_MSB_SENSOR;` – ustawia bufor na adres starszego bajtu danych.
  - `I2CWrite(LIGHT_I2C_ADDR, buf, 1);` oraz `I2CRead(LIGHT_I2C_ADDR, buf, 1);` – wysyłają i odczytują starszy bajt danych.
  - `data = (buf[0] « 8 | data);` – łączy młodszy i starszy bajt w jedną 16-bitową liczbę.
- **Przeliczenie wartości:**
  `return (range * data / width);` – przelicza wartość z czujnika na luksach, używając zakresu `range` i rozdzielczości `width`.

Włączanie i konfiguracja czujnika – funkcja `light_enable()`

Do aktywacji i konfiguracji czujnika służy funkcja:

```
void light_enable (void)
{
    uint8_t buf[2];
    buf[0] = ADDR_CMD;
    buf[1] = CMD_ENABLE;
    I2CWrite(LIGHT_I2C_ADDR, buf, 2);
    range = RANGE_K1;
    width = WIDTH_16_VAL;
}
```

Opis działania funkcji:

- `buf[0] = ADDR_CMD;` – stała o wartości 0x00, definiująca adres rejestru komendy w czujniku światła.
- `buf[1] = CMD_ENABLE;` – stała o wartości (1 « 7), reprezentująca komendę włączenia urządzenia.
- `I2CWrite(LIGHT_I2C_ADDR, buf, 2);` – wysyła przez I2C dwubajtowy pakiet (adres rejestru i komenda) na adres czujnika. Przyjmuje trzy argumenty: adres urządzenia I2C (LIGHT_I2C_ADDR), wskaźnik do bufora danych (buf), oraz liczbę bajtów do wysłania
- `LIGHT_I2C_ADDR` – (0x44 « 1) – adres I2C urządzenia, do którego chcemy wysłać dane/

- `range = RANGE_K1;` – ustawia zakres pracy urządzenia (np. 973), co definiuje czułość lub zakres pomiaru czujnika światła.
- `width = WIDTH_16_VAL;` – ustawia szerokość (np. 1 « 16), co oznacza 16-bitową rozdzielczość pomiaru.

Uwagi praktyczne i integracja sprzętowa

- ISL29003 jest stale podłączony do magistrali I2C i komunikuje się poprzez adres `LIGHT_I2C_ADDR` (0x44 « 1).
- Wyjście przerwania czujnika może być podłączone do pinu PIO2_5 mikrokontrolera. Przerwanie pozostaje aktywne (stan niski), dopóki nie zostanie zresetowane przez hosta za pomocą I2C.
- W celu ochrony linii sygnałowej przed uszkodzeniem, gdy sygnałem steruje więcej niż jeden sterownik, zastosowano szeregowy rezystor o wartości 270 Ω.
- Czujnik umożliwia szybkie i precyzyjne pomiary natężenia światła, co jest wykorzystywane m.in. do adaptacyjnego sterowania jasnością wyświetlacza lub jako źródło ziarna dla generatora losowego w aplikacji.

6. Analiza skutków awarii (FMEA)

Tabela 3. Analiza FMEA (Failure Mode and Effect Analysis).

Ryzyko	Prawdopodobieństwo	Znaczenie	(Samo)Wykrywalność	Reakcja	Iloczyn
Awaria timera	0.1	krytyczne (10)	Brak	Brak	1.0
Awaria dźwięku	0.1	niegroźne (2)	Brak	Brak	0.2
Awaria joysticka	0.3	krytyczne (10)	Brak	Brak	3.0
Awaria wyświetlacza	0.2	krytyczne (10)	Brak	Brak	2.0
Awaria linijki diod	0.1	niegroźne (2)	Brak	Brak	0.2
Awaria EEPROM	0.2	średnie (5)	Brak	Brak	1.0
Awaria akcelerometru	0.3	niegroźne (2)	Brak	Brak	0.6
Awaria czujnika oświetlenia	0.3	niegroźne (2)	Brak	Brak	0.6